Perancangan Arsitektur *Microservices* Untuk Resiliensi Sistem Informasi Perpustakaan Pusat (Studi Kasus UPN "Veteran" Jakarta)

Alif Garindra[1], Theresia Wati.[2], I Wayan Widi P.[3]
Jurusan Sistem Informasi, Fakultas Ilmu Komputer, UPN "Veteran" Jakarta
Jl. RS. Fatmawati Raya, Cilandak, Depok, Jawa Barat 12450
E-mail: garindra.alif@gmail.com[1], theresiawati@upnvj.ac.id[2], wayan.widi@upnvj.ac.id[3]

***Abstract** -- A good information system, besides providing convenience in serving information, it also needs to have a good resistant to disturbances. Lately, an architecture that is believed to be able to provide convenience in scalability, agility, resilience, manageability is growing. It known as microservice architecture, designed to share smaller and structured services to improve the quality of information systems. In the central library information system at the "Veteran" National Development University Jakarta, this architecture has not been used and is still using monolithic architecture. It is an architecture in which all components are made into a single unit. This means that the development, maintenance, and resilience of the system will be more difficult and time consuming because the frontend and backend coding processes are in the same service. To answer the problems that have been mentioned, the researcher wants to design a microservice architecture that will be used in the central library information system by using the waterfall method to create a microservice architecture, Node.js as a server maker, REST API as a standard data exchange and connection between services, Docker as a container and Mongodb as the NoSQL database.*

***Keywords:** Microservices Architecture, Node.js, RESTful API, Library, Mongodb*

Abstrak -- Sebuah sistem informasi yang baik, selain memberikan kemudahan dalam melayani informasi, namun juga tahan akan adanya gangguan. Akhir ini sedang marak sebuah arsitektur yang dipercaya dapat memberikan kemudahan dalam *scalability,agility,resiliency,manageability*. Arsitektur tersebut adalah arsitektur *microservice,* dirancang untuk membagi *service* lebih kecil dan terstruktur guna meningkatkan kualitas sistem informasi. Pada sistem informasi perpustakaan pusat di Universitas Pembangunan Nasional "Veteran" Jakarta, arsitektur ini belum digunakan dan masih menggunakan arsitektur monolitik. Yaitu sebuah arsitektur yang semua komponen dibuat menjadi sebuah kesatuan. Artinya untuk pengembangan, pemeliharaan, dan resiliensi dari sistem tersebut akan lebih sulit dan memakan waktu karena dalam proses pengkodingan *frontend* dan *backend* di satu *services* yang sama. Untuk menjawab permasalahan yang telah disebutkan, peneliti ingin merancang arsitektur *microservice* yang akan digunakan dalam sistem informasi perpustakaan pusat dengan menggunakan metode *waterfall* untuk membuat arsitektur *microservice, Node.js* sebagai pembuat *server, REST API* sebagai standar pertukaran data dan penghubung antar *services, Docker* sebagai *container* dan *Mongodb* sebagai *database* NoSQL.

***Kata Kunci:** Arsitektur Microservices, Node.js, RESTful API,* **Perpustakaan,** ***Mongodb*.**

I. PENDAHULUAN

Sebuah sistem informasi yang baik, selain memberikan kemudahan dalam melayani informasi, namun juga harus tahan akan adanya gangguan dan mampu beradaptasi. Baik dari segi pengembangan maupun pemeliharaan. Kemampuan menghadapi gangguan dan beradaptasi inilah yang dinamakan resiliensi.

Salah satu arsitektur yang kurang memiliki kemampuan resiliensi adalah arsitektur monolitik, dimana semua pengkodingan *backend* dan *frontend* dijadikan satu dalam *services* yang sama. Hal ini dapat menyebabkan ketergantungan yang erat antar setiap komponen. Sehingga jika terjadi kesalahan pada salah satu komponen akan mempengaruhi keseluruhan aplikasi dan jika ingin melakukan perubahan teknologi maka akan merubah keseluruhan aplikasi. Akhir ini muncul istilah arsitektur *microservices* sebagai arsitektur pengganti monolitik.

Arsitektur *microservices* adalah dimana sistem dirancang untuk membagi *service* lebih kecil dan terstruktur guna meningkatkan kualitas *software* di bidang resiliensinya. *Software* akan dirancang untuk menjalankan fungsi secara independen. Artinya, setiap permasalahan teknis dapat diselesaikan dengan cara dan teknologi yang berbeda beda yang nantinya akan dihubungkan dengan *Application Programming Interface (API).* [1]

Pada perpustakaan pusat Universitas Pembangunan Nasional "Veteran" Jakarta, ditemukan rancangan arsitektur monolitik pada *web services* nya, sehingga untuk pengembangan dan pemeliharaan akan sangat sulit. Salah satu cara untuk mengatasi permasalah tersebut adalah mengganti arsitekturnya dengan *microservices.*

Tujuan Penelitian

Berdasarkan pembahasan pada latar belakang diatas, tujuan peneliti merancang arsitektur *microservices* pada sistem informasi perpustakaan pusat ini untuk meningkatkan resiliensi dari sistem sebelumnya.

Penelitian Terdahulu

Peneliti mempunyai beberapa penelitian terdahulu sebagai referensi dan pedoman dalam proses dan me nyelesaikan penelitian ini. Dibawah ini terdapat tabel yang berisi tentang penelitian sebelumnya, hasil dari penelitian sebelumnya dan hubungan penelitian sebelumnya dengan penelitian ini.

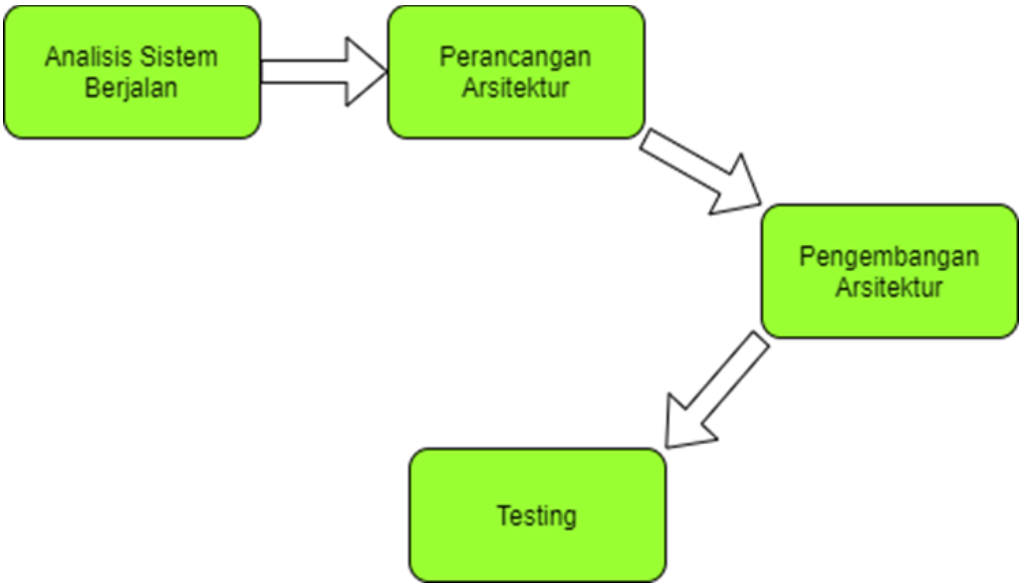
1. Penelitian yang dilakukan oleh Hatma Suryotrisongko pada tahun 2017 dengan judul "Arsitektur *Microservice* untuk Resiliensi Sistem Informasi". Penelitian ini membahas perihal cara meningkatkan resiliensi sistem informasi dengan menggunan arsitektur *microservice*. Sebenarnya penelitian ini merupakan lanjutan dari hasil penelitian sebelumnya, yaitu pengembangan *software* Open *Source* untuk manajemen asosiasi/keanggotaan, dengan tujuan untuk meningkatkan kualitas *software*. Arsitektur *microservice* dalam penelitian ini akan diimplementasikan ke dalam *web* AISINDO (Asosiasi Profesi Sistem Informasi Indonesia) dengan cara penelitian ulang kode program (*Refactoring*). Hasil dari penelitian ini menunjukkan implementasi *microservices* berhasil meningkatkan resiliensi sistem. [2]
2. Penelitian yang dilakukan oleh Rifki Mufrizal dan Dina Indarti pada tahun 2019 yang berjudul "*Refactoring* Arsitektur *Microservice* Pada Aplikasi Absensi PT". Graha Usaha Teknik" membahas tentang banyaknya arsitektur monolitik yang berkembang saat ini dan mempersulit kinerja, pemeliharaan dan kompleksnya pembaharuan aplikasi. Oleh karena itu penelitian ini menggunakan arsitektur *microservices* untuk mengatasi permasalahan tersebut khususnya pada aplikasi Absensi PT. Graha Usaha Teknik dengan cara *refactoring*. Pendekatan yang dilakukan penelitian ini menggunaka data kualitatif dan kuantitatif. Data kualitatif didapat dengan melakukan analisis terhadap proses bisnis dan arsitektur yang berjalan. Sedangkan data kuantitatif didapatkan dengan melakukan uji *response time* aplikasi terhadap *request* per satuan waktu. [3]
3. Penelitian yang dilakukan oleh Rahmad Ade Putra pada tahun 2019 yang berjudul "Analisa Implementasi Arsitektur *Microservices* Berbasis Kontainer Pada Komunitas Pengembang Perangkat Lunak Sumber Terbuka (*Opendaylight Devops Community*)". Penelitian ini bertujuan untuk menganalisa keberhasilan implementasi arsitektur microservices pada pengembangan perangkat lunak sumber terbuka *Opendaylight project*. Pengumpulan data berupa data kuantitatif dan kualitatif yang nantinya akan dijadikan variabel dalam uji coba perangkat lunak. Uji coba tersebut terdiri dari uji regresi *linear* berganda, uji korelasi berganda, uji koefisien determinasi, uji multikoleniaritas dan uji autokorelasi. Kesimpulan dari penelitian ini adalah arsitektur *microservices* memberikan layanan yang lebih cepat, sayangnya implementasi pada *Opendaylight* masih kurang dikarenakan masih banyak kriteria-kriteria yang belum maksimal. [4]

II. METODOLOGI PENELITIAN

Metodologi penelitian yang penulis gunakan adalah metode *waterfall* terdiri dari 4 tahap yaitu analisis sistem berjalan, perancangan arsitektur, pengembangan arsitektur, *testing*/pengujian. Pada tahap analisis sistem berjalan, dilakukan analisis berdasarkan observasi dan wawancara. Terdapat 2 hal yang peneliti Analisa, yaitu proses bisnis dan arsitektur sistem. Hasil analisis arsitektur akan digunakan dalam identifikasi masalah. Selanjutnya aka nada percancangan arsitektur, yaitu membuat permodelan setiap *services* yang akan dibuat dan *API Gateway*nya. Permodelan yang dilakukan termasuk diagram, tabel rincian tentang *database*, server, port dan teknik.

Ketika perancangan sudah terbuat, maka peneliti melakukan pengembangan arsitektur yaitu pengkodingan setiap *persistent microservices* menggunakan nodejs dan teknik *REST API*. Setelah itu peneliti menggunakan mongoose untuk menghubungkan nodejs dengan basis data *mongodb*. Setelah pengkodingan *persistent microservices* selesai barulah dilakukan kontainerisasi dengan *Docker* serta pembuatan *microservices* lainnya yang nanti akan saling dihubungkan menjadi *API Gateway*.

Tahap terakhir adalah pengujian resiliensi program, ada dua tipe pengujian yaitu pengujian mengatasi *downtime* dengan *runtime* dan pengujian untuk mempertahankan beberapa layanan pada saat terjadi kesalahan infrastruktur.



Gambar 1. Metodologi Penelitian

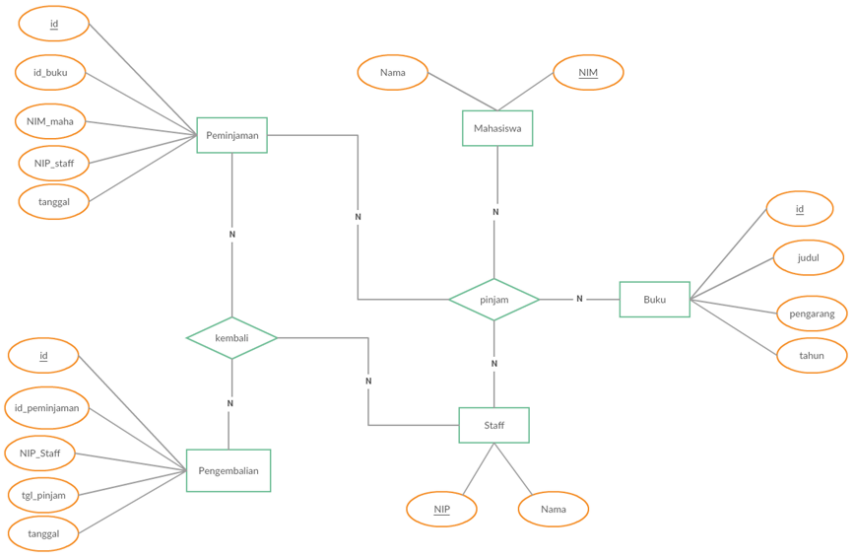
III. HASIL DAN PEMBAHASAN

Analisis Sistem Berjalan

Peneliti telah melakukan observasi dan wawancara di lingkungan Perpustakaan Pusat UPN "Veteran" Jakarta. Wawancara dilakukan dengan manajer dan staff perpustakaan pusat. Tujuan dari observasi dan wawancara yang berjalan ini sebagai referensi pembuatan arsitektur *microservices* agar tidak jauh dari proses bisnis yang berlangsung**.**

Dari hasil observasi dan wawancara, peneliti mengetahui bahwa proses bisnis peminjaman buku cukup baik karena sudah menggunakan *web application*. *Web* tersebut dibangun dengan, otomasi manajemen SLiMS (*Senayan Library Management System*) Akasia yang bersifat *open sources.*

Peneliti membuat *ERD* agar memudahkan peneliti dalam memahami hubungan antar entitas, selain itu *ERD* juga digunakan agar pemetaan entitas tidak melenceng dari sistem yang berjalan.



Gambar 2. ERD sistem berjalan

Gambar diatas merupakan *ERD* yang peneliti buat, penjabaran dari *ERD* tersebut adalah sebagai berikut.
1. Entitas
Entitas yang ada adalah mahasiswa, staff, buku, peminjaman dan pengembalian.
2. Relasi
a.Untuk melakukan peminjaman buku, entitas yang diperlukan adalah entitas mahasiswa, buku, dan staff. Yang nantinya *primary key* dari tiap entitas akan digunakan sebagai *foreign key* entitas peminjaman.
b.Untuk melakukan pengembalian buku, entitas yang diperlukan adalah entitas peminjaman dan staff. Yang nantinya *primary key* dari entitas peminjaman akan digunakan sebagai *foreign key* entitas pengembalian.
3. Attribut

Tabel 1. Atribut

No	Entitas	Attribut
1	Mahasiswa	NIM sebagai *primary key* Nama
2	Buku	Id sebagai *primary key* judul pengarang tahun
3	Staff	NIP sebagai *primary key* nama
4	Peminjaman	id sebagai *primary key* id_buku sebagai *foreign key* NIM_maha sebagai *foreign key* NIP_staff sebagai *foreign key* tanggal
5	Pengembalian	id sebagai *primary key* id_peminjaman sebagai foregn key NIP_staff sebagai *foreign key* tgl_pinjam tanggal

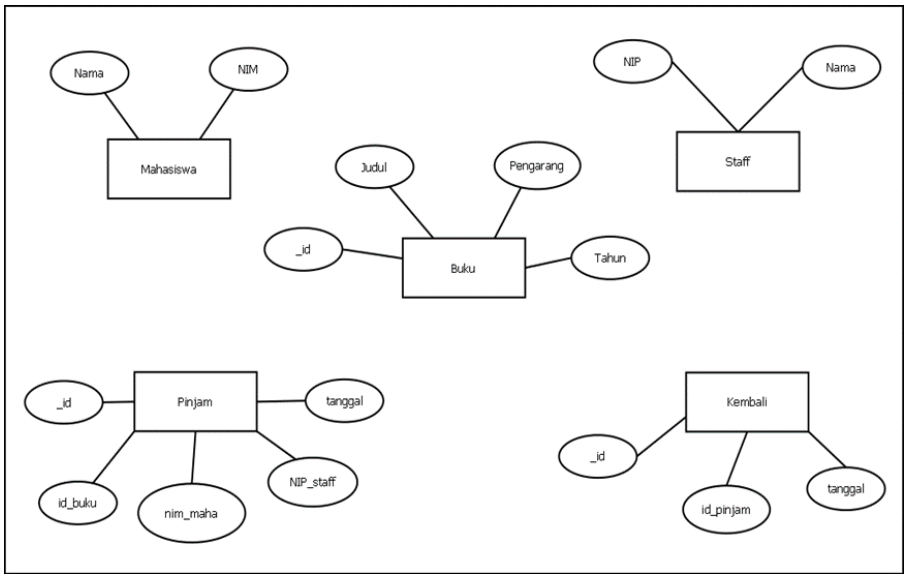
Analisis Proses Bisnis Sistem Berjalan -- Berdasarkan observasi langsung dan wawancara kepada staff perpustakaan pusat UPN "Veteran" Jakarta, proses bisnis dari sistem informasi perpustakaan puat UPNVJ sudah cukup baik dan peneliti jadikan sebagai referensi pembuatan sistem baru dengan arsitektur *microservices*.

Analisis Arsitektur Sistem Berjalan -- Peneliti menganalisis arsitektur dari portal Perpustakaan Pusat Universitas Pembangunan Nasional "Veteran" Jakarta dengan cara mengunduh *source code* dan mengkaji dokumentasi dari otomasi SLiMS Akasia. Peneliti menggunakan pendekatan kualitatif untuk menkaji arsitektur dari SLiMS Akasia. Setelah melakukan analisis, peneliti menemukan bahwa arsitektur dari SLiMS Akasia adalah arsitektur 3-tier dan *monolithic*.

Perancangan Sistem dengan Arsitektur *Microservices*

Peneliti berencana untuk membuat arsitektur *microservice* dengan referensi dari proses bisnis dari sistem yang berjalan. Tahap perancangan antara lain adalah penentuan dan pemetaan entitas, perancangan *microservices* pattern, perancangan *persistent microservices*, perancangan stateless *microservices*, perancangan *API Gateway* dan perancangan model domain.

Pemetaan Entitas -- Entitas yang akan peneliti buat berdasarkan refernsi dari sistem berjalan, namun tidak ada entitas bernama bidang akademis karena peneliti berfokus kepada proses bisnis peminjaman buku yang dilakukan pada proses berjalan. Entitas yang ada dapat dilihat seperti gambar dibawah.



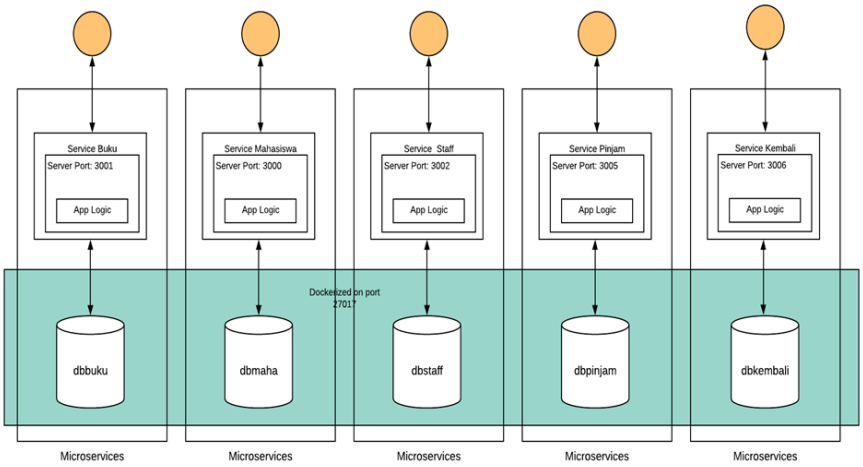
Gambar 3. Pemetaan Entitas

Dalam *Entity Relationship Diagram* sistem berjalan, entitas pinjam hanya memiliki attribut inti *id* peminjaman dan tanggal peminjaman, sisanya didapat dari attribut referensi. Namun pada perancangan entitas arsitektur *microservice*, attribut yang sebelumnya hanya sebuah attribut lemah yang didapat dari referensi seperti NIM mahasiswa, NIP *staff* dan *id* buku sekarang menjadi sebuah *id* inti yang lengkap. Maksud dari pembuatan attribut yang tadinya hanyalah referensi menjadi inti adalah memberikan keleluasaan pada pengembangan sebuah sistem. Untuk peneliti sendiri, entitas ini akan digunakan dalam pembangunan *persistence microservices*. Berdasarkan proses bisnis system berjalan, aktor yang memiliki *database* adalah staff dan mahasiswa.

Perancangan *Persistent Microservices* -- Peneliti memetakan entitas yang ada pada sistem sebelumnya menjadi *services* yang terpisah. Setiap layanannya akan memiliki *database* sendiri dan tidak mengetahui keberadaan service yang lain. Pemetaannya bisa dilihat di tabel berikut.

Tabel 2. *Persistent Microservices*

Service	**Fungsi**	***Server Engine***	**Pertukaran data**	***Database***
Mahasiswa	Melakukan *CRUD* kepada data mahasiswa.	*Node.js*	*REST API, JSON*	*Mongodb*
Staff	Melakukan *CRUD* kepada data staff.	*Node.js*	*REST API, JSON*	*Mongodb*
Buku	Melakukan *CRUD* kepada data buku.	*Node.js*	*REST API, JSON*	*Mongodb*
Pinjam	Melakukan *CRUD* kepada data peminjaman.	*Node.js*	*REST API, JSON*	*Mongodb*
Kembali	Melakukan *CRUD* kepada data pengembalian.	*Node.js*	*REST API, JSON*	*Mongodb*

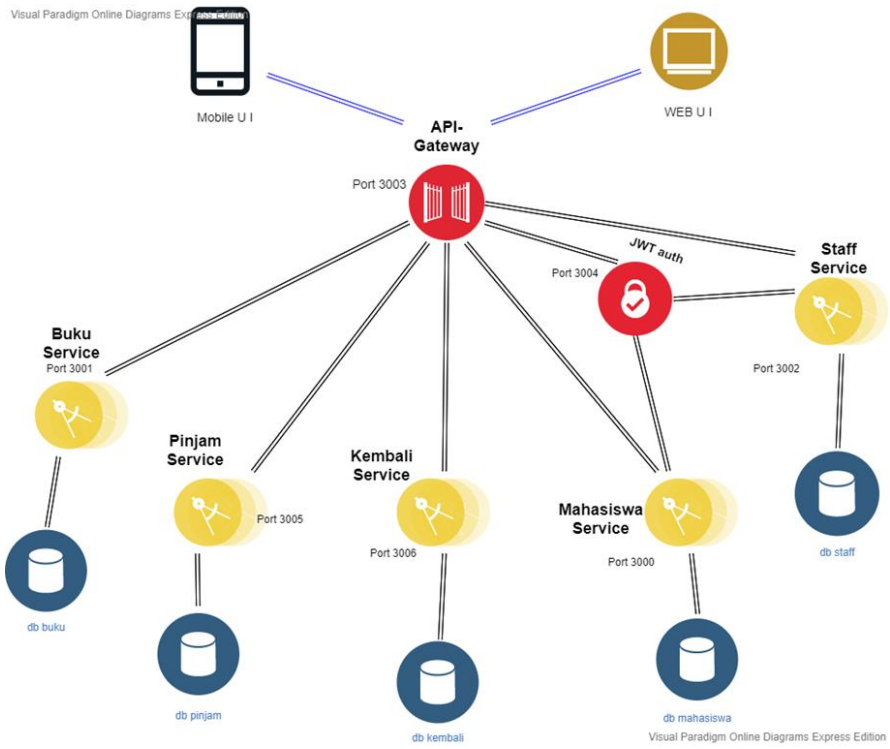


Gambar 4. Model *Persistent Microservices*

Berdasarkan gambar diatas, setiap *microservices* memiliki struktur yang sama persis. Setiap *service* memiliki *database*nya masing masing, setiap *services* harus berkomunikasi dengan HTTP Request berstandar *REST API*. Lalu setiap *database* dijalankan di dalam port yang sama yaitu 2701, dalam hal ini peneliti menggunakan *Docker* sebagai pembuat image virtual *database*. Selain *database* port pada tiap-tiap *services* akan berbeda, hal ini digunakan sebagai pemisah agar setiap *service* dapat dijalankan walaupun ada *service* lain yang sedang *down.*

Perancangan Stateless *Microservices* -- Pada perancangan Stateless *Microservices* peneliti memerlukan sebuah *microservices* baru yang memiliki fungsi untuk membuat logika keamanan kepada setiap *actor* yang terlibat dalam proses bisnis. Perancangannya berupa *server, application logic* dan *API Endpoint.*

Perancangan *API Gateway* dan Model Domain -- Agar semua *microservices* dapat dijadikan sebuah arsitektur yang lengkap dan dapat dikonsumsi oleh client maka diperlukan *API Gateway*. *API Gateway* yang peneliti buat selain berfungsi sebagai alat berkomunikasi antar *microservices* juga sebagai tempat autentifikasi dan autorisasi.



Gambar 5. *API Gateway* dalam Model Domain

API Gateway dirancang dengan menghubungkan port dari setiap *services* yang telah terisolasi dengan *container* menggunakan *REST API*, sehingga setiap *services* yang ada dapat dikonsumsi oleh *front end* dengan perantara *API Gateway*. Hal ini memungkinkan satu paket *backend* dapat digunakan beberapa platform tanpa harus merubah pengkodingan secara menyeluruh.

Pengembangan Arsitektur *Microservices*

Pada tahap ini peneliti mulai melakukan pengkodingan untuk membuat *service* yang saling terpisah. Tahapnya dapat dijabarkan sebagai berikut.

1. Pembuatan *server* aplikasi dari setiap layanan yang dibutuhkan, dengan *nodejs* untuk menjalankan *javascript* di sisi *server* lalu *expressjs* sebagai *framework* modul *HTTP*.
2. Pembuatan *database* dan *collection* dengan bantuan *mongoose* dan *image Docker* di *mongodb*.
3. Pembuatan *Application logic* dan *API Endpoint*
4. *Dockerizing* seluruh *server* yang ada sehingga bisa dijalankan secara bersamaan.
5. *API Documentation*

Pembuatan *Application* Server dan *Database* -- Pada tahap pertama pengembangan, peneliti membuat *server* dan *database* dari setiap *services* yang akan dibuat dengan menggunakan *expressjs*, *mongoose* dan *Docker*.

Pembuatan *Application Logic Persistent Microservices* -- Setiap *service* harus memiliki logika untuk melakukan *CRUD (Create, Read, Update, Delete)* terhadap *services*nya sendiri.

```
app.use(function(err,req,res,next){
    res.status(422).send({error: err.message})
});

app.get('/',function(req,res,next){
    res.json({msg:"ini service buku"});
});

app.get('/buku', function (req,res,next) {
    Buku.find({}).then(function(buku){
        res.send(buku);
    }).catch(next)
});

app.post('/buku', function (req,res,next) {
    Buku.create(req.body).then(function(buku){
        res.send(buku);
    }).catch(next);
});
```

Gambar 6. Potongan Kode Logika Aplikasi

Jika disimpulkan maka potongan kode diatas berarti server meminta sebuah data dari dalam *collection*, jika sudah maka akan diberikan *response* berupa data berformat *json*, namun jika gagal akan tertampil *error*. Peneliti menggunakan metode ini untuk pembuatan semua *persistent microservices* kecuali yang berfungsi sebagai aktor. Untuk *persistent microservices* yang juga memiliki fungsi sebagai aktor peneliti menambahkan *Bcrypt* agar *password actor* tersebut dapat disimpan dalam bentuk yang telah di enkripsi.

Pembuatan *Application Logic*Stateless *Microservices* -- Stateless *microservices* yang peneliti buat adalah *jwt auth* yang berfungsi untuk memberikan autentifikasi dan autorisasi kepada para akor untuk melakukan perintah sesuai dengan *role*-nya masing masing.

Pembuatan *API Gateway* -- Pembuatan *API Gateway* menggunakan *resources* dari *services* yang telah peneliti buat sebelumnya. Agar dapat terhubung secara cepat maka peneliti memutuskan untuk melakukan pemanggilan secara *asynchronous* dengan metode *fetch*.

```
//untuk login

app.post('/login/mahasiswa',(req,res)=>{
    const {nim,password} = req.body
    fetch('http://auth:3004/mahasiswa/login', {
        method: "POST",
        headers: {
            "Content-Type":"application/json"
        }, body: JSON.stringify({nim,password})})

            .then(response => response.json())
            .then(json => res.json(json))
            .catch(err => console.log(err));
})

app.post('/login/staff',(req,res)=>{
    const {nip,password} = req.body
    fetch('http://auth:3004/staff/login', {
        method: "POST",
        headers: {
            "Content-Type":"application/json"
        }, body: JSON.stringify({nip,password})})

            .then(response => response.json())
            .then(json => res.json(json))
            .catch(err => console.log(err));
})
```

Gambar 7. Potongan Kode *Index API Gateway*

Pada potongan kode diatas, peneliti melakukan beberapa pemanggilan *fetch* ke *microservices* lain karena setiap *microservices* tidak diperbolehkan untuk saling berhubungan. Potongan kode diatas adalah bagian dari *index API Gateway* yang artinya setiap *actor* dapat mengaksesnya tanpa harus ter-autentifikasi.

Dockerizing -- Pada penelitian ini, *Docker* membantu peneliti untuk menjalankan semua *microservices* dan *database* yang memiliki *port* berbeda beda. Peneliti menambahkan *Dockerfile* pada tiap tiap folder *microservices* yang berfungsi untuk membuat *virtual image* dari setiap *microservices*. Setelah setiap *services* memiliki *virtual image*-nya masing-masing, selanjutnya dibuatlah *Docker-compose* agar setiap *virtual image* dapat dijalankan secara bersamaan.

API Documentation -- *API documentation* adalah sebuah cara mendokumentasikan setiap perincian yang berkenaan dengan *API* yang peneliti kerjakan. Mulai dari perincian *input*, proses hingga *output* yang harusnya diterima oleh *API*. Peneliti menjabarkan dokumentasi berdasarkan *API Gateway*.

Gambar 8. Contoh *API Gateway*

Peneliti menggunakan request pengambilan semua buku dari *API Gateway* sebagai contoh dokumentasi tanpa autentifikasi. Pemanggilan dengan metode *GET* yang dilakukan terhadap *port* 3003 atau *port* buku, memiliki hasil sukses tanpa *error* dengan menampilkan buku-buku yang telah tersimpan dalam *database mongodb*.

Pengujian Resiliensi

Pengujian *Runtime* -- Pada pengujian *runtime* peneliti berfokus untuk melihat performa sistem, dengan cara memasukkan beberapa http request pada saat yang bersamaan kedalam *API Gateway* untuk melihat apakah terjadi perbedaan *runtime* dari setiap *request* yang diolah.

Tabel 3. Hasil Uji *Runtime*

Label	# Samples	Average	Median	Min	Max	Error %	Throughpu	Received K	Sent KB/se
tampilkan	20	8	6	5	32	0.00%	20.85506	14.75	3.26
lihat semu	20	1736	1640	269	3485	0.00%	4.55373	4.14	1.61
lihat semu	20	1689	1742	268	3654	0.00%	4.34311	4.68	1.54
staff login	20	3119	3128	633	4209	0.00%	3.9557	1.7	0.85
Mahasiswa	20	3167	3359	948	4244	0.00%	3.94322	1.76	0.89
TOTAL	100	1944	2024	5	4244	0.00%	19.71609	14.07	5.15

Semua waktu melakukan *request* dilakukan dalam waktu yang sama, namun sampel pengambilan data semua buku selesai pertama, sedangkan *login* mahasiswa dan *staff* selesai terakhir. Hal ini disebabkan karena pengambilan data pada buku diambil dari *persistent microservices* milik buku, tanpa ada pengaksesan lain. Sedangkan segala proses *login* selesai terakhir karena dua faktor, yaitu ada *request* pengambilan data dari *persistent microservices* dan proses perubahan data menjadi *token* dari *stateless microservices auth* yang dilakukan secara bersamaan.

Hal ini berguna bagi pengembang *frontend* dalam hal resiliensi dan performa. Dapat dikatakan menambah resiliensi dan performa sistem karena satu halaman pada *User Interface* dapat dipecah menjadi beberapa pemanggilan yang tidak saling membebani satu sama lain sehingga dapat meminimalisir terjadinya *server down*.

Microservices Failure Simulation -- Pada pengujian ini peneliti berfokus pada resiliensi sistem. Peneliti akan melakukan simulasi seakan akan salah satu *microservices* mengalami kesalahan, dan melihat apakah akan mengganggu *microservices* lain. Peneliti mematikan salah satu *server* dari *microservices* yaitu *server* mahasiswa sebagai simulasi terjadinya kesalahan pada infrastruktur sistem.

Tabel 4. Hasil *Microservices Failure Simulation*

Label	# Samples	Average	Median	Min	Max	Error %	Throughpu	Received K	Sent KB/se
tampilkan	50	291	332	68	481	0.00%	38.63988	27.32	6.04
lihat semu	50	2713	2900	329	4803	0.00%	8.67152	9.34	3.08
staff login	50	3801	3722	1188	5784	0.00%	7.47719	3.21	1.61
Mahasiswa	50	8772	8764	8283	9266	100.00%	5.39433	13.46	0
lihat semu	50	8736	8732	8246	9233	100.00%	5.41477	13.52	0
TOTAL	250	4863	4319	68	9266	40.00%	26.92805	38.8	3.91

Pada hasil pengujian ditemukan bahwa kerusakan salah satu *service* tidak mempengaruhi *services* yang lain. Pemeliharaan akan lebih mudah di dilakukan karena setiap kegagalan pada sebuah layanan dapat di lokalisir dengan mudah. Misalnya terjadi kegagalan pada layanan mahasiswa, maka *developer* akan segera mengetahui bahwa masalah terdapat pada *microservices* mahasiswa, sehingga dapat mempercepat pengambilan keputusan dan pemeliharaan

V. KESIMPULAN

Sistem Informasi Perpustakaan Pusat Universitas Pembangunan Nasional "Veteran" Jakarta, masih memiliki arsitektur monolitik yang minim akan resiliensi. Maka peneliti melakukan perancangan dan pembuatan arsitektur *microservices* pada perpustakaan pusat dengan menggunakan metode *waterfall*. Hasil dari pengujian arsitektur *microservices* yang telah peneliti rancang berhasil meningkatkan resiliensi Sistem Informasi Perpustakaan Pusat UPN "Veteran" Jakarta. Resiliensi yang telah berhasil ditingkatkan adalah mengurangi *downtime* dan mempertahankan beberapa layanan yang ada walau terjadi kesalahan infrastruktur, hal tersebut membantu *developer* untuk melokalisir kesalahan dan melakukan pemeliharaan sistem.

DAFTAR PUSTAKA

[1] R. Y. Pratama, "microservices apaan tuh," 11 October 2019. [Online]. Available: https://medium.com/codelabs-unikom/microservices-apaan-tuh-b9f5d56e8848.
[2] H. Suryotrisongko, "Arsitektur Microservice untuk Resiliensi Sistem Informasi," Jurnal Sisfo Vol. 06 No. 02, pp. 236-248, 2017.
[3] R. &. D. I. Mufrizal, "Refactoring Arsitektur Microservice Pada Aplikasi Absensi PT. Graha Usaha Teknik.," Jurnal Nasional Teknologi dan Sistem Informasi Vol. 05 No. 01, pp. 58-67, 2019.
[4] R. A. Putra, "ANALISA IMPLEMENTASI ARSITEKTUR MICROSERVICES BERBASIS KONTAINER PADA KOMUNITAS PENGEMBANG PERANGKAT LUNAK TERBUKA (OPENDAYLIGHT DEVOPS COMMUNITY)," JUST IT Vol. 9 No. 2, pp. 150-162, 2019.